Note: Problem 3 is a little challenging! If you get everything working fully, your maximum grade on this homework is 45/35 points.

1. **(5 pts)** Write a recursive method **compareStrings(String s1, String s2)**. This method should return 0 if the strings **s1** and **s2** contain exactly the same characters in the same order, 1 if **s1** is alphabetically (using ASCII values) “greater than” **s2**, or -1 if **s1** is alphabetically “less than” **s2**. Hint: Think about the base case(s)! When can you immediately return a value without needing any recursive calls?

Examples:

**compareStrings(“Enterprise”, “defiant”)** should return -1 (since ASCII values for capitals precede those for lower-case characters)

**compareStrings(“enterprise”, “enter”)** should return 1 **compareStrings(“Excelsior”, “”)** should return 1 **compareStrings(“”, “Excelsior”)** should return -1

1. **(15 pts)** In the **RecursiveLL<E>** class that we discussed in lecture, add recursive methods for each of the following wrappers. Your recursive methods should add at least one parameter to keep track of the current node in the list. You can use the posted code as a starting point.
   1. *(4 pts)* **contains(E someItem)** – returns **true** if **someItem** exists in the list, **false** if not.
   2. *(4 pts)* **search(E someItem)** – returns the index of the first occurrence of **someItem** if it exists in the list, or -1 if **someItem** is not in the list.
   3. *(7 pts)* **reverse()** – reverses the calling list. (Hint: write a recursive method that reverses the list starting from a specific node, returning the head of the reversed list.)
2. **(25 pts)** You are developing the next great graphics program to compete with the likes of Adobe Photoshop and GIMP. At the moment, your program works only on binary images (i.e., each pixel can be only black or white).

You are working on a “magic wand” feature where clicking on a single pixel will select that pixel as well as all contiguous pixels. Two pixels *A* and *B* are considered contiguous if 1) they are the same color, and 2) they can be connected by a path of pixels that have the same color as *A* and *B*. Diagonal pixels are not considered contiguous. For example, in the image below, pixels *A* and *B* are contiguous, but *A* and *C* are not.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  | *A* |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | *C* |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | *B* |  |  |

Write a program that allows the user to specify a text file to read image data from. Each row of pixels consists of 0’s and 1’s separated by spaces, with 0 indicating white and 1 indicating black. You can assume that each row of the text file will have the same number of columns. However, the number of rows and columns can be anything. For example, the image above would be represented in the text file as

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **1** | **1** | **1** | **0** | **0** | **1** | **1** | **1** |
| **1** | **1** | **1** | **1** | **0** | **0** | **0** | **1** | **1** |
| **1** | **1** | **1** | **1** | **0** | **0** | **0** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** | **0** | **1** | **1** | **0** |
| **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **0** |
| **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** |
| **0** | **0** | **1** | **1** | **0** | **0** | **1** | **0** | **0** |

Once the image data is loaded, display the data using the space character for a white pixel and the character **.** for a black pixel. For example, the image above would be displayed as

**.... ...**

**.... ..**

**.... .**

**..... ..**

**........**

**.. .**

**.. .**

The user should then be able to select the row and column to start the magic wand operation from, and the program should display the selected pixels using the **\*** character. Here are some examples using the image above:

Starting from row 0, column 0:

**\*\*\*\* ...**

**\*\*\*\* ..**

**\*\*\*\* .**

**\*\*\*\*\* \*\***

**\*\*\*\*\*\*\*\***

**\*\* \***

**\*\* \***

Starting from row 0, column 7:

**.... \*\*\***

**.... \*\***

**.... \***

**..... ..**

**........**

**.. .**

**.. .**

Starting from row 5, column 0:

**.... ...**

**.... ..**

**.... .**

**..... ..**

**........**

**\*\*.. .**

**\*\*.. .**